High Frequency Magnetic Loop Antenna Auto-Tuner for Amateur Radio

Gregory Raven KF5N

May 2022

Contents

Introduction

This is the manual and ”user guide” for a magnetic loop antenna controller which is patterned after a design published in the book ”Microcontroller Projects for Amateur Radio” by Jack Purdum W8TEE and Albert Peter AC8GY. The author’s firmware for the antenna controller was developed in the Arduino IDE.

This derivative design replaces the STM32 ”Blue Pill” controller module with a Raspberry Pi Pico. I noticed the Pi Pico was very similar in size to the STM32 board, and had the necessary GPIOs and ADC (Analog to Digital Converter) to perform the same function as the Blue Pill board. Another reason to use the Pi Pico is that it has not been affected by “supply chain” problems which have plagued other electronic endeavors.

The firmware was developed in the Visual Studio Code text editor which is provisioned by a script provided by Raspberry Pi. This script installs and configures everything required to develop firmware for the Pi Pico. The C/C++ “Software Development Kit” (SDK) provided by Raspberry Pi has a wealth of functionality to control the peripheral components of the Pi Pico.

One of the prime reasons to chose the Pi Pico is that the development kit includes a “debugger”, which is a feature the Arduino IDE lacks. However, this may change as the Arduino project migrates to the 2.0 IDE.

As the original project was developed as an Arduino project, a lot of “hacking” of the code was required to adapt it to the Raspberry Pi SDK. The first experiment was to hack the display library since this was the most complex library in the project. After a few days of experimenting with both the library code and Cmake, the hacked display library was demonstrated functioning! The other libraries were much easier to translate to the Pi Pico SDK.

A future version of firmware may be provided which once again uses the Arduino IDE.

Although this project follows the same pattern as the original, there are other significant hardware changes:

- Replaced large external stepper driver with A4988 stepper driver module. The module is much smaller and is integrated into the PCB.
- Replaced MC1458 op amps with LM358. This eliminates the requirement for a negative supply voltage, which in turn eliminates the negative DC supply module.
- A BNC RF connector is added.
- An Ethernet connector is added. This is used to connect the controller to the antenna tuner motor and limit switches.
- A coaxial power connector is added for the 12 volt power required.
- Electronic switches were added to save standby power consumption.
- A +12 Volt relay control output was added.
- The number of pushbuttons required was reduced from seven to three. This was allowed due to simplification of the user interface.
- An audio muting board can be added to mute speaker audio during the autotuning process.
- A shunt diode plus fuse style reverse polarity protection was added.

The buttons and menu system are de-bounced.

The PCB is 100mm by 100mm and can be fabricated for approximately US $1 (not including shipping cost).

The enclosure is 3D printed and is similar to the design of the original project.

The controller should function with the original project's "double-double" loop antenna. I decided to go a simpler route with a basic single-loop design. The loop is resonated with an eBay sourced butterfly capacitor. The antenna uses a combination of PVC plumbing and 3D printed parts.

The remainder of this document should provide an experienced homebrewer with enough information to duplicate the controller and antenna. The Github repositories containing the project information will be periodically updated as the firmware and components continue to evolve.

Hardware

0.0.1 Microcontroller

The original design used an STM32-based board commonly known as the ”Blue Pill”. This is a single-core ARM microcontroller with 64kb of Flash memory, and 20kb of RAM. It has an Analog to Digital Converter (ADC) which is required by the SWR measuring function of the project. The board is supported in the Arduino IDE and has a large range of supporting libraries.

A drawback of the ”Blue Pill” in the Arduino IDE is that there is more than one source for the supporting libraries. These libraries are not necessarily interchangeable. Configuration of the IDE and libraries can be confusing.

Compounding the ”Blue Pill” problem is the numerous variants of the board with microcontrollers sourced from various vendors. There are geniune ST Micro devices, unauthorized clones, and authorized clones. The officially supported Arduino support software will reject the unauthorized clone devices. Alternative support libraries may or may not work with the clones.

When you buy a ”Blue Pill”, you may not be able to determine exactly what you are getting.

In early 2021 Raspberry Pi made a surprise announcement of a new microcontroller product called the ”Pi Pico”. The microcontroller is a dual-core ARM system-on-chip with an ADC, and enough general purpose input-output (GPIO) to support the magnetic loop controller. Flash memory is a much larger 2MB and 260kB of RAM.

The Pi Pico form factor is quite similar to the Blue Pill. The price is also similar, with an advertised cost of $4 (not including shipping). Pi Pico boards have been consistently available in spite of the semiconductor supply chain issues seen in recent months.

Making the Pi Pico greatly attractive is the expansive and well-supported Software Development Kit (SDK) along with numerous examples. The downside of this is new device is that there are not many libraries available, although the ecosystem is continuing to expand.

Arduino support is available, however, this author has not explored this option yet. This will be further discussed in the firmware section.

0.0.2 DDS Module, RF Amplifier and SWR Bridge

The DDS Module, RF Amplifier, and VSWR bridge circuits are direct copies of the original project.

The RF amplifier consumes significant current, so it was decided to add a PMOS switch connected to a GPO so that the amplifier can be deactivated when not required.

One quirk of the SWR bridge circuit is that it is sensitive to the forward voltage drop of the 1N5711 diodes used. This will not affect the ability of the controller to tune to minimum SWR, but it will affect the accuracy of the SWR reported on the display. I found better results when the forward voltage drop of the 1N5711 diodes is lower. Diodes ordered from Amazon by “Chanzon” performed the best.

0.0.3 Stepper Motor Driver

The original project used a large external stepper motor driver called the TB6600. This is a very nice device with DIP switches to control step size and current drive. It is capable of 1/32 microsteps. The main problem with this device is that it is larger than the entire PCB and needs to stand up vertically. So the project's case must be much larger to accommodate this stepper driver.

Meanwhile, I had taken the cover off the controller of my 3D printer, and discovered that it used very small plug-in stepper driver boards. These driver boards are about the size of a postage stamp! They are based on the A4988 driver device. However, these drivers are capable of only 1/16 microsteps. This requires an external DIP switch or resistors to select. The current is adjusted by a very tiny potentiometer on the board. The authors of the original project stated they used 1/16 microsteps, so perhaps this small board would be adequate. So far it has! The size reduction of the finished device in the case is substantial. The A4988 cost is about $2, versus $12 for the TB6600, so it is a nice cost reduction as well. The modules can be purchased from Amazon or eBay.

0.0.4 External RF Relay

In order for the controller to tune the antenna to resonance and minimize SWR, it must be directly connected to the antenna. After tuning, the antenna must be switched over to the RF transceiver. An electromagnetic relay is required. A PMOS switch is used in the controller to switch +12 Volts to a coaxial power connector on the rear of the controller. This should be connected to the coil of the RF relay. The coil is de-energized during normal operation of the transceiver. The coil is energized only when the controller is auto-tuning. Thus when controller power is off, the transceiver is connected to the antenna.

0.0.5 Audio Muting

Please refer to the chapter on the audio muting board.

Schematic

0.1 Schematic

The RF circuitry from the DDS to the output of the SWR bridge was a direct copy of the original project.

The forward and reverse SWR amplifiers were changed significantly. These two-stage operational amplifiers have sufficient gain to amplify the SWR bridge output to a level high enough to utilize close to the full input range of the microcontroller's ADCs. Another function of these amplifiers is to linearize the output of the SWR bridge, which is a pair of diode detectors. A diode feedback circuit is used in the input stage, which linearizes and improves the dynamic range of the SWR measuring system.

The original system used 741 operational amplifiers with a dual +/-5volt supply. This requires a negative 5 volt power supply board.

The 741 op-amps are replaced by LM358s, which can operate with a single supply and have a useful input range to ground. Since the SWR detectors output positive voltage only, this is acceptable. These op-amps have low input offset voltage, however, it is enough to disrupt the SWR measurement accuracy. Input offsets are effectively removed in software. The circuit is very similar to the original with the exception of the single supply replacing the dual.

Two PMOS transistors controlled by GPOs are added to reduce current drain during stand-by. The +5 Volt PMOS switch also provides a signal to the audio muting board during auto-tune. A third PMOS outputs +12 Volts to an RF relay to switch the antenna to the controller during auto-tune.

A 1N4004 diode is in shunt with the +12 Volt DC power input. This diode is used with a fuse mounted in the rear panel for reverse-polarity protection.

An A4988 stepper motor driver module is added. This replaces the large external motor driver used in the original project. The A4988 module is connected to an Ethernet connector on the rear of the PCB. Limit switch inputs are also routed to the Ethernet connector.

The Raspberry Pi Pico module was wired according to the directions in the documentation:

https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf

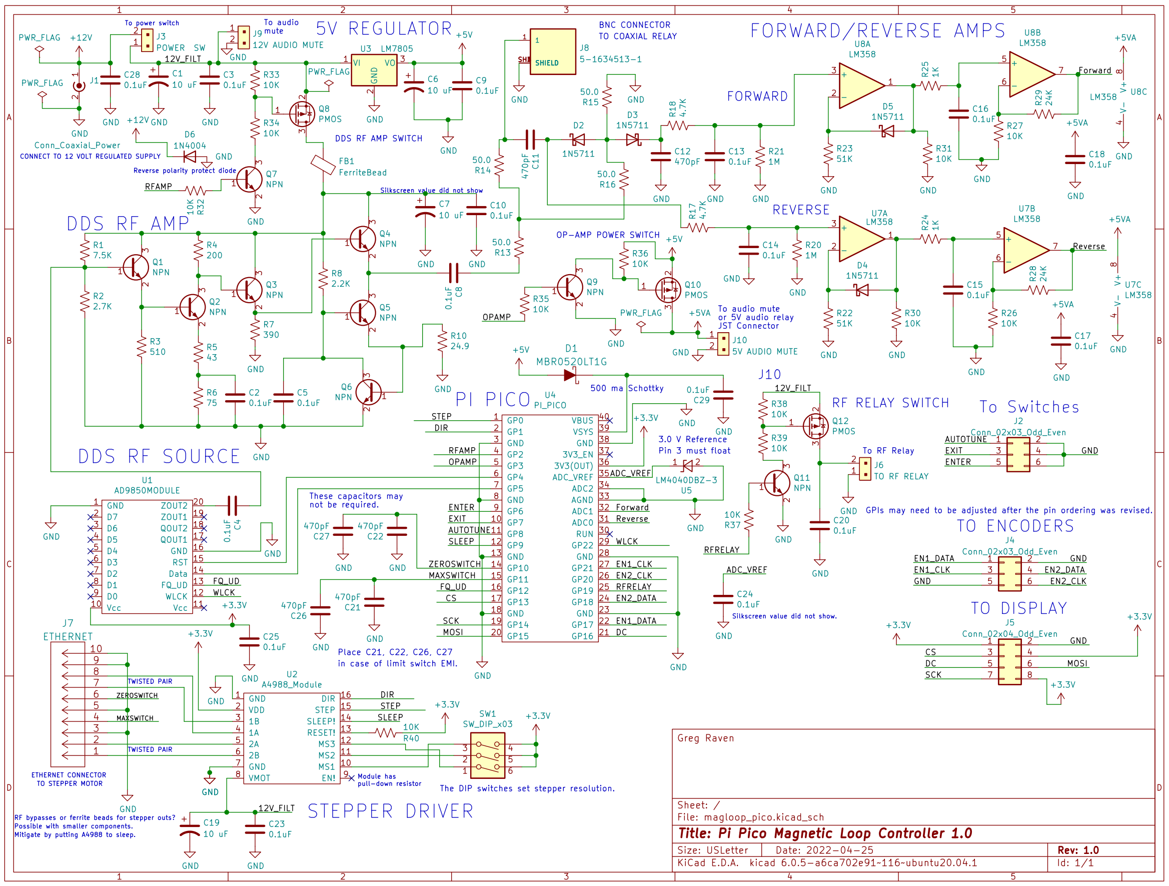

5V REGULATOR
FORWARD/REVERSE AMPS
FORWARD
REVERSE
DDS RF AMP
DDS RF AMP SWITCH
OP-AMP POWER SWITCH
DDS RF SOURCE
PI PICO
RF RELAY SWITCH
To Switches
TO ENCODERS
TO DISPLAY
STEPPER DRIVER
ETHERNET
BNC CONNECTOR TO COAXIAL RELAY
CONNECT TO 12 VOLT REGULATED SUPPLY
Reverse polarity protect diode
Silkscreen value did not show
500 ma Schottky
3.0 V Reference
Pin 3 must float
To audio mute or 5V audio relay JST Connector
These capacitors may not be required.
Place C21, C22, C26, C27 in case of limit switch EMI.
GPIs may need to be adjusted after the pin ordering was revised.
ETHERNET CONNECTOR TO STEPPER MOTOR
RF bypasses or ferrite beads for stepper outs? Possible with smaller components. Mitigate by putting A4988 to sleep.
The DIP switches set stepper resolution.
Module has pull-down resistor
Greg Raven
Sheet: /
File: magloop_pico.kicad_sch
Title: Pi Pico Magnetic Loop Controller 1.0
Size: USLetter
Date: 2022-04-25
Rev: 1.0
KiCad E.D.A. kicad 6.0.5-a6ca702e91-116-ubuntu20.04.1
Id: 1/1

Figure 1: Magnetic Loop Controller Schematic

Printed Circuit Board

0.1.1 Printed Circuit Board

The first version PCB was derived from the original STM32 project. Since the Pi Pico is approximately the same size, it was dropped in place.

The first Pi Pico board used a through-hole wire terminal for the stepper motor and limit switch connections. These are connections which must be made through a long cable to the remote antenna tuner.

Cheap lawn sprinker cable was used to connect the controller board to the tuner via the terminal strip. This would function fine with a short cable. However, upon trying a 50 foot long cable the controller went nuts!

What happened is that the very square pulses output from the stepper driver were coupling to the limit switch inputs. This causes big problems as the controller senses an incoming pulse train on the limit switch inputs, rather than a single step which is what should happen when a limit switch closes.

So I took this problem to the Groups.io Software Controlled Amateur Radio group, which is frequented by builders and users (and the authors of the book)of the original magnetic loop controller. Similar problems were seen, with the best solution to change to a shielded CAT7 Ethernet cable.

After successful testing breadboards of the Ethernet cable "fix" for the limit switch problem, it was decided to make a new PCB with an Ethernet connector mounted to the board.

This version of the board added PMOSFET switches for an antenna relay, the 5 volt supply to the op-amps, and 12 volt supply to the RF amplifier. Unfortunately there was only one spare GPIO, so all of these switches operated simultaneously.

The new external RF relay switch is required to disconnect the controller from the antenna during normal transmit/receive operation after autotuning is complete. The other switches are mainly intended for current drain reduction when the circuits are not required in normal operation.

The second prototype board was a success, however, the Ethernet connector was located on the side of the board. This means an Ethernet jumper would be required to connect to another Ethernet connector on the rear of the case. Not good!

So it was decided to fundamentally rearrange the main components on the PCB for the third prototype. The Ethernet connector was moved to the rear of the board, adjacent to the power connector. This worked well, as the stepper driver was also moved to the back and this made the routing of 12 volts to the stepper driver much shorter.

Both the Pi Pico and the DDS boards were moved to new locations. When the re-routing was complete, the board routing was simplified, and it should perform better than the second prototype.

Routing of signals to the IDC connectors for display, encoders, and pushbuttons was done such that making the ribbon cables is as easy as possible.

Firmware

The firmware for the project was developed using a Raspberry Pi 4. Raspberry Pi has done an outstanding job providing a script which installs everything required to develop code for the Pi Pico, including Visual Studio Code, which is an advanced text editor. It is essentially an IDE.

Debugging is done via Serial Wire Debugger interface (SWD). The SWD is connected to the Pi Pico via three jumpers from the Raspberry Pi 4 GPIO connector to a three pin header on the Pi Pico. No special "dongle" is required!

The supported debugger is the main reason this project was developed using the SDK. Debugging is only recently available in the Arduino IDE 2.0. However, an initial look at the Arduino debugger indicated it was not mature compared to the Pi Pico SDK. In the future, the Arduino IDE may be revisited if the Pi Pico support includes a solid debugging feature.

0.1.2 Organization of the Code

The code is organized differently than the original project. The original project is an Arduino style project, with the use of C++ classes (libraries) and multiple files with associated C-style functions in each. It is the typical "setup" and "loop" structure of Arduino.

This new project based on the Pi Pico is more strongly C++. Rather than files with numerous C-style functions, these functions were gathered up and put into C++ classes. The main function instantiates these classes in the form of several objects. I found this "C++ embedded programming" helped me to create easier to understand code. The fundamental programming principle is called "separation of concerns":

https://en.wikipedia.org/wiki/Separation_of_concerns

"Inheritance" was used where it naturally fell into place. For example, the "StepperManagement" functions were organized into a C++ class, and this class inherits from the AccelStepper class library.

The "state machine" pattern was used in several places including the GUI menu in the main function.

The CMake build system is used to compile, link, and create various associated files. This is one disadvantage of using the Pi Pico SDK. You don't have to deal with this in Arduino. After a bit of struggle with CMake, I got it all working together smoothly.

0.1.3 Data Class

There is a new "Data" class which stores all of the constants used to configure the controller. This class is intended to replace all or most of the #define statements which appear in the original design. A data object is instantiated in the main program at start-up. The data object is referenced by the other objects and is used to access the relevant constants.

0.1.4 EEPROM Class

The EEPROM class performs the same function as the original project. It provides non-volatile storage for the band limit stepper motor positions after initial calibration. Like the original project, it uses FLASH memory to mimic EEPROM functionality. The Pi Pico SDK FLASH functions were used to create this class.

0.1.5 ILI9341 Driver

As mentioned before, the library support for the Pi Pico is in the early stages. I needed a library for driving the ILI9341 TFT display. There is an Arduino driver, and I set off on a mission to hack it to work with the Pi Pico.

The problem with hacking an Arduino library is that the code is designed to work with multiple architectures. The code is littered with #if statements which select blocks of code depending on which architecture is in use. This was painful to work with, on top of the fact that it is a complex driver.

After hours and hours of hacking, I got the ILI9341 working with the Pi Pico. If a driver specifically intended for the Pi Pico appears, I will certainly consider replacing the monstrous hack I created. Having stated that, it has been working reliably and consistently with all of the prototypes. Having the display is also great for debugging because you can print anything you want to the display. That combined with the debugger made the rest of the code development quite pleasant.

The original Arduino library:

`https://github.com/adafruit/Adafruit_ILI9341`

0.1.6 Stepper Motor Library

The original "AccelStepper" library was modified to work with the Pi Pico. The modifications for this library were minimal.

The original Arduino library:

`https://www.airspayce.com/mikem/arduino/AccelStepper/`

0.1.7 Rotary Encoder Library

The original "Rotary" library was modified to work with the Pi Pico. The modifications for this library were minimal.

The original Arduino library:

`https://github.com/buxtronix/arduino/tree/master/libraries/Rotary`

0.1.8 De-bouncing

De-bouncing functions are included in the Button class.

The original project used simple time-delays for debouncing the buttons. This project has state-machine based de-bounce. This was challenging to implement, as it also had to include de-bounce when moving up and down the menu levels. An auxiliary function was required along with careful placement of the debounce functions in the code.

0.1.9 Limit Switch Protection

The low and high limit switches are connected to GPIs via the CAT7 cable. These GPI states are continuously monitored during stepper motor movements. If either switch closes, the stepper motor will be stopped. This is to protect the limit switches from damage due to over-rotation of the stepper motor. This is most likely to happen during "bring up" of the controller as connections could be accidentally reversed. It is also possible to hit the high limit if the controller is commanded to go to a frequency beyond the limit of the upper limit of antenna resonance.

In case of an upper limit event, the user is notified of an "Upper Limit Hit" via the display. Note that the primary function of the low limit switch is for calibration of the stepper motor. Its secondary function is to protect itself from destruction.

0.2 Git Repositories

Move this to the appendix

Loop Antenna Construction

0.2.1 3D Printed Parts

images of loop antenna here

0.2.2 Butterfly Capacitor

images of butterfly capacitor here

0.2.3 Capacitor Protective Enclosure

images of enclosure here

Using Ethernet Cable for Stepper Motor Connection

The controller works by rotating a stepper motor connected to a tuning capacitor. As the antenna approaches resonance, there is a sharp “dip” in the SWR, which is continually monitored by the controller via the antenna coaxial cable.

Thus it is required to drive a stepper motor from the controller to whatever remote location the antenna is located.

The challenge with this is that the stepper motor drive is a series of sharp pulses! This means the cable connecting the stepper motor to the controller must not have excessive resistance, parasitic inductance, and capacitance. Fortunately, the pulses are very low frequency so it does not have to be perfect. I was skeptical at first that such remote operation of a stepper would work reliably, but this scheme has proven robust enough to rotate the variable capacitor. The fact that this is a low-torque application of a stepper motor helps make it possible.

In addition to the four wires for the stepper motor, there must be wires for two “limit switches”. There is a “low” switch and a “high” switch. Both of these switches are used to protect from over-rotating the capacitor. More importantly, the low switch is important for calibration of the system. The controller will automatically “zero” out the stepper when the controller is powered on. The zero step point coincides with the fully-meshed, maximum capacitance setting of the variable capacitor. All further operations of the controller are dependent on the zero step point being accurately established at power on.

A major problem can occur as the limit switch wires are bundled in the same cable as the stepper wires. Stray capacitance in the wires causes the sharp pulses to be coupled to the low and high limit switch wires, which are connected to the Pi Pico’s GPIs. This will cause “falsing” of the limit switch detect, and the result is erratic operation of the controller.

At first, I was using cheap utility wire from the local home improvement store. The pulses on the limit switch wires were extreme, so I decided to consult with other users of the controller who frequent the Software Controller Ham Radio group (groups.io). Some were using capacitors on the limit switches, and others were using Ethernet cables. The Ethernet cable idea is suggested in the book, but somehow I didn’t notice that.

I wasn’t familiar with how Ethernet cable is constructed, so I took at look at Wikipedia:

After looking at the various forms of Ethernet cable, I decided to try Cat 7. This has shielded twisted pairs, which will minimize cross-talk. There is also a shield around the entire cable assembly which can be grounded.

There are a total of four twisted pairs. This is perfect, as the stepper motor signals can be connected to two of the twisted pairs, and the limit switches can be connected to the other two.

Note that the shield around the entire cable assembly should be grounded. This requires an Ethernet connector which makes this connection. This connector must be carefully selected.

The stepper driver module is located adjacent to the Ethernet connector to allow short and simple routing. The Ethernet cable plugs into the back of the controller.

Note that shunt capacitors were added to the board in case of limit switch problems. I haven't detected problems so far, however, my Ethernet cable is only 50 feet in length. Others have reported successful operation with cables of 100 feet.

An Ethernet connector to screw terminal adapter board is used at the antenna. This board is available from Amazon or eBay. The BOM has a link to this board. An example 50 foot length Cat7 cable is also in the BOM. The cable is claimed to be waterproof and outdoor rated.

images of ethernet cables and connectors here

Controller Enclosure

The enclosure components are 3D printed. I'm a beginner in 3D printing, so what I did was print out some of the components of the original controller enclosure, which were published as STL files. I have adapted these designs for this new controller, and published the design files in FreeCad and STL format. The builder should probably optimize these files for their particular printer using a "slicer" application like Cura.

I have used nylon hardware for assembling the enclosure, although typical steel or brass hardware should be OK. There is a good selection of nylon hardware in a kit available from Amazon. A link to this kit is in the BOM.

The enclosure was adjusted for minimal convenient size to fit the PCB and display. The rear panel has several holes for the various connectors exiting the rear of the controller.

There are four holes in the base for attaching silicone feet. You will need to drill holes in the adhesive silicone feet to allow them to be attached with bolts. The adhesive is not strong enough for this application. The feet will not stay in place. The feet were purchased from Amazon. A link to them is in the BOM.

ADD COMMENTS ABOUT TOP COVER HERE

include enclosure photo(s) here

User Interface

The user interface is both simplified and expanded compared to the original project. The original project includes a total of seven pushbuttons: five discrete pushbuttons, and two buttons integrated into the encoders.

The force required to push the encoder buttons is enough to make the controller slide around. Those were the first priority for elimination.

Changing the menu system to use “Enter” and “Exit” buttons eliminated two more buttons. The only remaining dedicated button is “AutoTune”, which is central to the mission of the controller.

include user interface diagrams here

Speaker Audio Muting

One annoying problem became apparent during system testing of the controller with antenna and radio attached. The architecture of this controller is that it provides the RF signal for measuring SWR during auto-tuning. This is the purpose of the AD9850 DDS module.

However, this large signal is going to be heard in the receiver if it is tuned to the desired frequency! It is a very loud tone, and although brief, it is very irritating.

The solution to this problem is simple, but it requires additional circuitry. There was not enough room to implement this on the controller board, so an auxiliary PCB was designed to take care of this function.

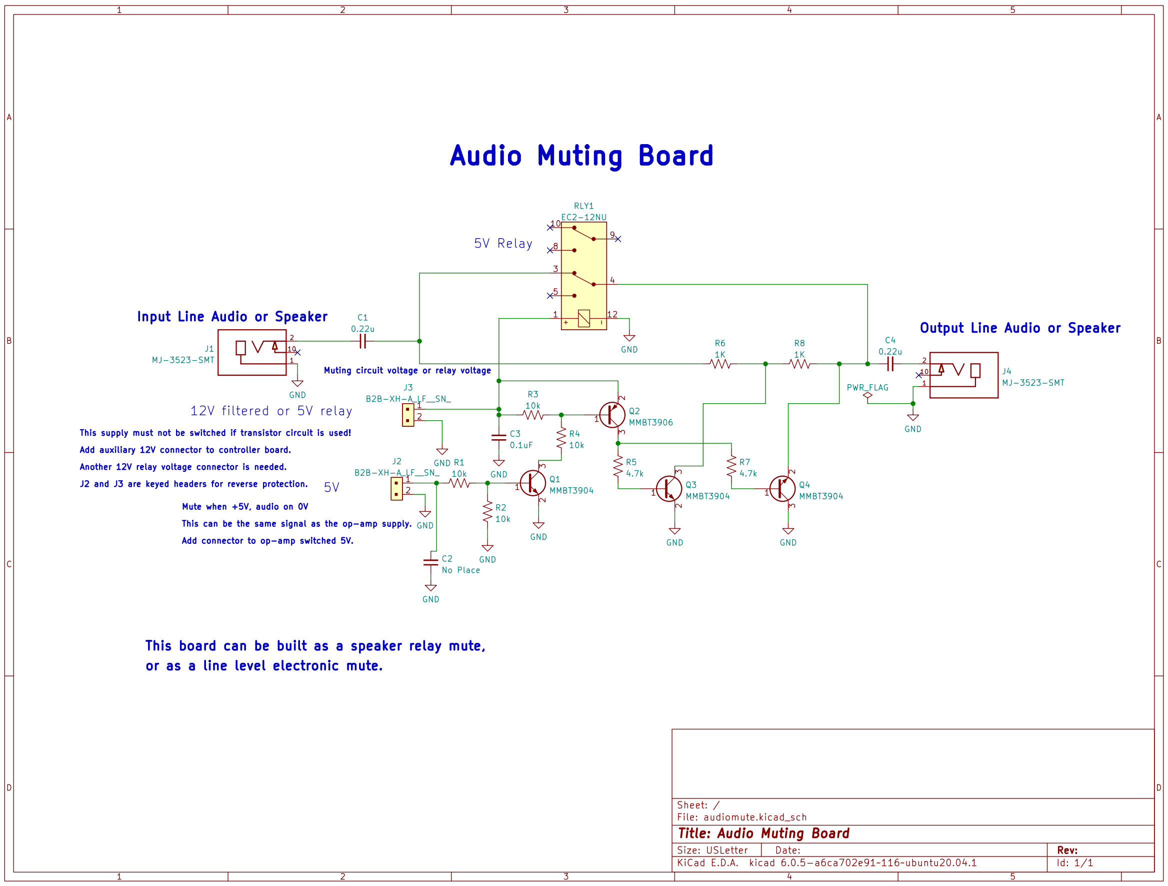
The circuit functions by simply muting the receiver audio during auto-tuning. The audio muting PCB is designed so that it can be built as a "line level" muting circuit or a simple electromagnetic relay to connect-disconnect the speaker output.

I didn't know the simplest way to mute audio (besides a relay), but an internet search yielded this web page:

`https://www.sound-au.com/articles/muting.html`

The line-level muting circuit is based on information from this web page. The simple relay circuit is sort of intertwined with the line-level circuit, but it should be easy to see how to build the board depending on the user's preference.

The board requires two simple two-wire cables to connect to the controller board. Because these connections have power and ground, the connectors had to be polarized. JST type connectors were used.



This supply must not be switched if transistor circuit is used!
Add auxiliary 12V connector to controller board.
Another 12V relay voltage connector is needed.
J2 and J3 are keyed headers for reverse protection.

Mute when +5V, audio on 0V
This can be the same signal as the op−amp supply.
Add connector to op−amp switched 5V.

This board can be built as a speaker relay mute,
or as a line level electronic mute.

Figure 2: Audio Muting Board Schematic

Construction

In this chapter I will throw out a few ideas to make the assembly of the controller faster and easier.

0.2.4 PCB Assembly

First, get all of the parts ready and reasonably sorted. Get all of the surface-mount resistors and capacitors into a bin, semiconductors in a bin, the leaded parts in another bin, the modules in another, etcetera.

It is a good idea to assemble everything on a grounded static mat. I did not adhere to this 100% of the time and nothing has failed yet, although latent defects can show up later. Just try to keep from zapping your parts, OK?

In general, you should start with the lowest height parts and then the next height, and the next, etcetera.

Start with the passive surface-mount parts. These are all very low height on the board, and it is easier to solder them without taller leaded parts obstructing the soldering iron. Get all of the surface-mount resistors and capacitors soldered to the PCB. Use a good clean fine-point soldering iron time, and fine solder.

Up next are the transistors and diodes. Be especially careful with the Schottky diode which routes power to the Pi Pico. The cathode marking is hard to see, so you might want to check with the diode function on a VTVM to make sure you are orienting the diode in the correct direction. The SOT-23 NPN and PMOS transistors are marked on the PCB. There is also a SOT-23 3.0 volt regulator device which is located underneath the Pi Pico.

Next, solder the four (DO35) Schottky detector diodes, making sure to insert with the correct polarity ("K" on the PCB means cathode).

Next are the through-hole LM358 operational amplifiers and DIP SW1.

Next are the JST connectors. Be sure to make the orientation of these connectors consistent with respect to the ground pin. These cables carry supply voltage, so do not cross-wire them to ground!

Next, there are six headers for the modules. Be sure to get these soldered in straight, as this makes insertion of the modules easier.

Next, the IDC connectors for the display, switches, and pushbuttons.

Next, the BNC and coaxial power connector. The BNC connector has a lot of thermal mass, so you should change to a larger soldering iron tip. You may need to use a larger soldering iron, or use two soldering irons.

Next, the four leaded electrolytic capacitors, with care to insert them with the correct polarity.

Next, the Ethernet connector, and make sure the ground tabs are properly soldered.

Next, the LM7805 regulator, making sure it is inserted with the correct orientation. Add an aluminum heatsink to the LM7805.

0.2.5 Ribbon Cables

images of latest and greatest ribbon cables here

0.2.6 JST Cables

image of JST cable crimper here images of JST cables here

0.2.7 Power and Fuse Wiring

Zoom of power and fuse wiring